A. Proof of Theorems

A.1. Proof of Theorem 4.1

To prove Theorem 4.1, we need the following key lemma:

Lemma A.1. *For any 1-Lipschitz function $f(\boldsymbol{x})$ (with respect to ℓ_∞ norm) on a bounded domain $\mathbb{K} \in \mathbb{R}^n$, and any $\epsilon > 0$, there exists a finite number of functions $f_i(\boldsymbol{x})$ such that for all $\boldsymbol{x} \in \mathbb{K}$*

$$\max_i f_i(\boldsymbol{x}) \le f(\boldsymbol{x}) \le \max_i f_i(\boldsymbol{x}) + \epsilon,$$

where each $f_i(\boldsymbol{x})$ has the following form

$$f_i(\boldsymbol{x}) = \min_{1\le j\le n} \{x_j - w_j^{(i)}, w_j^{(i)} - x_j\} + b_i.$$

Proof of Lemma A.1. Without loss of generality we may assume $\mathbb{K} \in [0,1]^n$. Consider the set $\mathbb{S}$ consisting of all points $\frac{\epsilon}{2}(N_1, ..., N_n)$ where N_j are integers, we can write $\mathbb{S} \cap \mathbb{K} = \{\boldsymbol{w}^{(1)}, ..., \boldsymbol{w}^{(m)}\}$ since it's a finite set. $\forall \boldsymbol{w}^{(i)} \in \mathbb{S} \cap \mathbb{K}$, we define the corresponding $f_i(\boldsymbol{x})$ as follows

$$\begin{aligned} f_i(\boldsymbol{x}) &= \min_{1\le j\le n} \{x_j - w_j^{(i)}, w_j^{(i)} - x_j\} + f(\boldsymbol{w}^{(i)}) \\ &= -\|\boldsymbol{x} - \boldsymbol{w}^{(i)}\|_\infty + f(\boldsymbol{w}^{(i)}) \end{aligned} \tag{7}$$

On the one hand, we have $f(\boldsymbol{x}) - f_i(\boldsymbol{x}) = f(\boldsymbol{x}) - f(\boldsymbol{w}^{(i)}) + \|\boldsymbol{x} - \boldsymbol{w}^{(i)}\|_\infty \ge -\|\boldsymbol{x} - \boldsymbol{w}^{(i)}\|_\infty + \|\boldsymbol{x} - \boldsymbol{w}^{(i)}\|_\infty = 0$, therefore $\forall \boldsymbol{x} \in \mathbb{K}$ we have $f(\boldsymbol{x}) \ge f_i(\boldsymbol{x})$, namely $\max_i f_i(\boldsymbol{x}) \le f(\boldsymbol{x})$ holds.

On the other hand, $\forall \boldsymbol{x} \in \mathbb{K}$ there exists its 'neighbour' $\boldsymbol{w}^{(j)} \in \mathbb{S} \cap \mathbb{K}$ such that $\|\boldsymbol{x} - \boldsymbol{w}^{(j)}\|_\infty \le \frac{\epsilon}{2}$, therefore by using the Lipschitz properties of both $f(\boldsymbol{x})$ and $f_j(\boldsymbol{x})$, we have that

$$\begin{aligned} f(\boldsymbol{x}) &\le f(\boldsymbol{w}^{(j)}) + \frac{\epsilon}{2} = f_j(\boldsymbol{w}^{(j)}) + \frac{\epsilon}{2} \\ &\le f_j(\boldsymbol{x}) + \epsilon \le \max_i f_i(\boldsymbol{x}) + \epsilon \end{aligned} \tag{8}$$

Combining the two inequalities concludes our proof. □

Lemma A.1 "decomposes" any target 1-Lipschitz function into simple "base functions", which will serve as building blocks in proving the main theorem. We are ready to prove Theorem 4.1:

Proof of Theorem 4.1. By Lemma A.1, there exists a finite number of functions $f_i(\boldsymbol{x})$ $(i = 1, ..., m)$ such that $\forall \boldsymbol{x} \in \mathbb{K}$

$$\max_i f_i(\boldsymbol{x}) \le \tilde{g}(\boldsymbol{x}) \le \max_i f_i(\boldsymbol{x}) + \epsilon \tag{9}$$

where each $f_i(\boldsymbol{x})$ has the form

$$f_i(\boldsymbol{x}) = \min_{1\le j\le n} \{x_j - w_j^{(i)}, w_j^{(i)} - x_j\} + \tilde{g}(\boldsymbol{w}^{(i)}) \tag{10}$$

The high-level idea of the proof is very simple: among width $d_{\text{input}} + 2$, we allocate d_{input} neurons each layer to keep the information of $\boldsymbol{x}$, one to calculate each $f_i(\boldsymbol{x})$ one after another and the last neuron calculating the maximum of $f_i(\boldsymbol{x})$ accumulated.

To simplify the proof, we would first introduce three general basic maps which can be realized at a single unit, then illustrate how to represent $\max_i f_i(\boldsymbol{x})$ by combing these basic maps.

Let's assume for now that any input to any unit in the whole network has its ℓ_∞-norm upper bounded by a large constant C, we will come back later to determine this value and prove its validity.

Proposition A.2. *$\forall j, k, p$ and constant w, c, the following base functions are realizable at the kth unit in the lth hidden layer:*

1, the projection map:

$$u(\boldsymbol{x}^{(l)}, \theta^{(l,k)}) = x_j^{(l)} + c \tag{11}$$

2, the negation map:

$$u(\boldsymbol{x}^{(l)}, \theta^{(l,k)}) = -x_j^{(l)} + c \tag{12}$$

3, the maximum map:

$$\begin{aligned} u(\boldsymbol{x}^{(l)}, \theta^{(l,k)}) &= \max\{x_j^{(l)} + w, x_p^{(l)}\} + c \\ u(\boldsymbol{x}^{(l)}, \theta^{(l,k)}) &= \max\{-x_j^{(l)} + w, x_p^{(l)}\} + c \end{aligned} \tag{13}$$

Proof of Proposition A.2. 1, the projection map: Setting $u(\boldsymbol{x}^{(l)}, \theta^{(l,k)})$ as follows

$$u(\boldsymbol{x}^{(l)}, \theta^{(l,k)}) = \|(x_1^{(l)}, ..., x_j^{(l)} + 2C, ..., x_n^{(l)})\|_\infty - 2C + c$$

2, the negation map: Setting $u(\boldsymbol{x}^{(l)}, \theta^{(l,k)})$ as follows

$$u(\boldsymbol{x}^{(l)}, \theta^{(l,k)}) = \|(x_1^{(l)}, ..., x_j^{(l)} - 2C, ..., x_n^{(l)})\|_\infty - 2C + c$$

3, the maximum map: Setting $u(\boldsymbol{x}^{(l)}, \theta^{(l,k)})$ as follows

$$\begin{aligned} &u(\boldsymbol{x}^{(l)}, \theta^{(l,k)}) \\ =&\|(x_1^{(l)}, ..., x_j^{(l)} + w + 2C, ..., x_p^{(l)} + 2C, ..., x_n^{(l)})\|_\infty \\ &- 2C + c \end{aligned}$$

$$\begin{aligned} &u(\boldsymbol{x}^{(l)}, \theta^{(l,k)}) \\ =&\|(x_1^{(l)}, ..., x_j^{(l)} - w - 2C, ..., x_p^{(l)} + 2C, ..., x_n^{(l)})\|_\infty \\ &- 2C + c \end{aligned}$$

We finish the proof of Proposition A.2. □

With three basic maps in hand, we are prepared to construct our network. Using proposition A.2, the first hidden layer realizes $u(\boldsymbol{x}, \theta^{(1,k)}) = x_k$ for $k = 1, ..., d_{\text{input}}$. The rest two units can be arbitrary, we set both to be x_1.

By proposition A.2, throughout the whole network, we can set $u(\boldsymbol{x}^{(l)}, \theta^{(l,k)}) = x_k$ for all l and $k = 1, ..., n$. Notice that $f_i(\boldsymbol{x})$ can be rewritten as

$$\begin{aligned} f_i(\boldsymbol{x}) = -\max\{&x_1 - w_1^{(i)}, \max\{w_1^{(i)} - x_1, \\ &\max\{..., w_n^{(i)} - x_n\}...\}\} + \tilde{g}(\boldsymbol{w}^{(i)}) \end{aligned} \tag{14}$$

Using the maximum map recurrently while keeping other units unchanged with the projection map, we can utilize the unit $u(\boldsymbol{x}^{(l)}, \theta^{(l,d_{\text{input}}+1)})$ to realize one $f_i(\boldsymbol{x})$ at a time. Again by the use of maximum map, the last unit $u(\boldsymbol{x}^{(l)}, \theta^{(l,d_{\text{input}}+2)})$ will recurrently calculate (initializing with $\max\{f_1(\boldsymbol{x})\} = f_1(\boldsymbol{x})$)

$$\max_i f_i(\boldsymbol{x}) = \max\{f_m(\boldsymbol{x}), \max\{..., \max\{f_1(\boldsymbol{x})\}...\}\} \tag{15}$$

using only finite depth, say L, then the network outputs $g(\boldsymbol{x}) = u(\boldsymbol{x}^{(L)}, \theta^{(L,1)}) = u(\boldsymbol{x}^{(L-1)}, \theta^{(L-1,d_{\text{input}}+2)}) = \max_i f_i(\boldsymbol{x})$ as desired. We are only left with deciding a valid value for C. Because $\mathbb{K}$ is bounded and $\tilde{g}(\boldsymbol{x})$ is continuous, there exists constants C_1, C_2 such that $\forall \boldsymbol{x} \in \mathbb{K}$, $\|\boldsymbol{x}\|_\infty \le C_1$ and $|\tilde{g}(\boldsymbol{x})| \le C_2$, it's easy to verify that $C = 2C_1 + C_2$ is valid. □

A.2. Proof of Theorem 4.2

We prove the theorem in two steps. One step is to provide a margin bound to control the gap between standard training error and standard test error using Rademacher complexity. Next step is to bound the gap between test error and robust test error. We will first give a quick revisit on Rademacher complexity and its properties, then provide two lemmas corresponding to the two steps for the proof.

Rademacher Complexity Given a sample $X_n = \{\boldsymbol{x}_1, ..., \boldsymbol{x}_n\} \in \mathbb{K}^n$, and a real-valued function class $\mathbb{F}$ on $\mathbb{K}$, the Rademacher complexity of $\mathbb{F}$ is defined as

$$R_n(\mathbb{F}) = \mathbb{E}_{X_n}\left(\frac{1}{n}\mathbb{E}_\sigma\left[\sup_{f\in\mathbb{F}}\sum_{i=1}^n \sigma_i f(\boldsymbol{x}_i)\right]\right)$$

where σ_i are drawn from the Rademacher distribution independently, i.e. $\mathbb{P}(\sigma_i = 1) = \mathbb{P}(\sigma_i = -1) = \frac{1}{2}$. It's worth noting that for any constant function r, $R_n(\mathbb{F}) = R_n(\mathbb{F}\oplus r)$ where $\mathbb{F}\oplus r = \{f + r | f \in \mathbb{F}\}$.

Rademacher complexity is directly related to generalization ability, as shown in Lemma A.3:

Lemma A.3. *(Theorem 11 in Koltchinskii et al. (2002)) Let $\mathbb{F}$ be a real-valued hypothesis class. For all $t > 0$,*

$$\mathbb{P}\left(\exists g \in \mathbb{F} : \mathbb{E}_{(\boldsymbol{x},y)\sim\mathcal{D}}\left[\mathbb{I}_{yg(\boldsymbol{x})\le 0}\right] > \Phi(g)\right) \le 2e^{-2t^2}.$$

where

$$\Phi(g) = \inf_{\delta\in(0,1]}\left[\frac{1}{n}\sum_{i=1}^n \mathbb{I}_{y_i g(\boldsymbol{x}_i)\le\delta} + \frac{48}{\delta}R_n(\mathbb{F}) + \left(\frac{\log\log_2(\frac{2}{\delta})}{n}\right)^{\frac{1}{2}}\right] + \frac{t}{\sqrt{n}}.$$

Lemma A.3 can be generalized to the following lemma:

Lemma A.4. *Let $\mathbb{F}$ be a real-valued hypothesis class, define the r-margin test error $\beta_r(g)$ as: $\beta_r(g) = \mathbb{E}_{(\boldsymbol{x},y)\sim\mathcal{D}}\left[\mathbb{I}_{yg(\boldsymbol{x})\le r}\right]$, then for any $t > 0$,*

$$\mathbb{P}\left(\exists g \in \mathbb{F} : \beta_r(g) > \Phi_r(g)\right) \le 2e^{-2t^2}.$$

where

$$\Phi_r(g) = \inf_{\delta\in(0,1]}\left[\frac{1}{n}\sum_{i=1}^n \mathbb{I}_{y_i g(\boldsymbol{x}_i)\le\delta+r} + \frac{48}{\delta}R_n(\mathbb{F}) + \left(\frac{\log\log_2(\frac{2}{\delta})}{n}\right)^{\frac{1}{2}}\right] + \frac{t}{\sqrt{n}}.$$

Proof of Lemma A.4. To further generalize Lemma A.3 to $\beta_r(g)$ with $r > 0$, we use the fact that Rademacher complexity remain unchanged if the same constant r is added to all functions in $\mathbb{F}$. Lemma A.4 is a direct consequence by replacing m_f by $m_f - r$ at the end of the proof of Theorem 11 in (Koltchinskii et al., 2002), where it plugs m_f into Theorem 2 in (Koltchinskii et al., 2002). □

Also, it's well known (using Massart's Lemma) that Rademacher complexity can be bounded by VC dimension:

$$R_n(\mathbb{F}) \le \sqrt{\frac{2VCdim(\mathbb{F})\log\frac{en}{VCdim(\mathbb{F})}}{n}} \tag{16}$$

We next bound the the gap between test error and robust test error. We have

Lemma A.5. *Assume function g is 1-Lipschitz (with respect to ℓ_∞-norm. The r-robust test error $\gamma_r(g)$ is no larger than the r-margin test error $\beta_r(g)$, i.e., $\gamma_r(g) \le \beta_r(g)$.*

Proof of Lemma A.5. Since $g(\boldsymbol{x})$ is 1-Lipschitz, then $yg(\boldsymbol{x}) > r$ implies that $\inf_{\|x'-x\|_\infty\le r} yg(\boldsymbol{x}') > 0$, therefore

$$\mathbb{E}_\mathcal{D}\left[\mathbb{I}_{yg(\boldsymbol{x})>r}\right] \le \mathbb{E}_\mathcal{D}\left[\inf_{\|x'-x\|_\infty\le r}\mathbb{I}_{yg(\boldsymbol{x}')>0}\right],$$

which implies $1 - \beta_r \le 1 - \gamma_r$. □

Now we are ready to prove Theorem 4.2. Based on Lemma A.4, Lemma A.5 and Eqn. 16, it suffices to bound the VC dimension of an ℓ_∞-dist net. We will bound the VC dimension of an ℓ_∞-dist net by reducing it to a fully-connected ReLU network. We first introduce the VC bound for fully-connected neural networks with ReLU activation borrowed from (Bartlett et al., 2019):

Lemma A.6. *(Theorem 6 in (Bartlett et al., 2019)) Consider a fully-connected ReLU network architecture F with input dimension d, width $w \geq d$ and depth (number of hidden layers) L, then its VC dimension satisfies:*

$$VCdim(F) = \tilde{O}(L^2 w^2) \tag{17}$$

The following lemma shows how to calculate ℓ_∞-distance using a fully-connected ReLU network architecture.

Lemma A.7. *$\forall \boldsymbol{w} \in R^d$, there exists a fully-connected ReLU network h with width $O(d)$ and depth $O(\log d)$ such that $h(\boldsymbol{x}) = \|\boldsymbol{x} - \boldsymbol{w}\|_\infty$.*

Proof of Lemma A.7. The proof is by construction. Rewrite $\|\boldsymbol{x}-\boldsymbol{w}\|_\infty$ as $\max\{x_1 - w_1, w_1 - x_1, ..., x_d - w_d, w_d - x_d\}$ which is a maximum of $2d$ items. Notice that $\max\{x, y\} = \mathrm{ReLU}(x-y) + \mathrm{ReLU}(y) - \mathrm{ReLU}(-y)$, therefore we can use $3d$ neurons in the first hidden layer so that the input to the second hidden layer are $\max\{x_i - w_i, w_i - x_i\}$, in all d items. Repeating this procedure which cuts the number of items within maximum by half, within $O(\log d)$ hidden layers this network finally outputs $\|\boldsymbol{x} - \boldsymbol{w}\|_\infty$ as desired. □

The VC bound of ℓ_∞-dist Net is formalized by the following lemma:

Lemma A.8. *Consider an ℓ_∞-dist net architecture F with input dimension d, width $w \geq d$ and depth (number of hidden layers) L, then its VC dimension satisfies:*

$$VCdim(F) = \tilde{O}(L^2 w^4) \tag{18}$$

Proof of Lemma A.8. By Lemma A.7, each neuron in the ℓ_∞-dist net can be replaced by a fully-connected ReLU subnetwork with width $O(w)$ and depth $O(\log w)$. Therefore a fully-connected ReLU network architecture G with width $O(w^2)$ and depth $O(L \log w)$ can realize any function represented by the ℓ_∞-dist net when parameters vary. Remember that VC dimension is monotone under the ordering of set inclusion, we conclude that such ℓ_∞-dist Net architecture F has VC dimension no more than that of G which equals $\tilde{O}(L^2 w^4)$ by lemma A.6. □

Finally, Theorem 4.2 is a direct consequence by combing Lemmas A.4, A.5 and A.8.

Remark A.9. *Though there exist generalization bounds for general Lipschitz model class (von Luxburg & Bousquet, 2004), the dependence on n scales as $n^{-1/d}$ which suffers from the curse of dimensionality (Neyshabur et al., 2017). Theorem 4.2 is dimension-free instead.*

B. Interval Bound Propagation

We now give a brief description of IBP (Mirman et al., 2018; Gowal et al., 2018), a simple convex relaxation method to calculate the certified radius for general neural networks. The basic idea of IBP is to compute the lower bound and upper bound for each neuron layer by layer when the input $\boldsymbol{x}$ is perturbed.

Input layer. Let the perturbation set be an ℓ_∞ ball with radius ϵ. Then for the input layer, calculating the bound is trivial: when x is perturbed, the value in the i-th dimension is bounded by the interval $[x_i - \epsilon, x_i + \epsilon]$.

Bound propagation. Assume the interval bound of layer l has already been obtained. We denote the lower bound and upper bound of layer l be $\underline{\boldsymbol{x}}^{(l)}$ and $\overline{\boldsymbol{x}}^{(l)}$ respectively. We mainly deal with two cases:

- $\boldsymbol{x}^{(l)}$ is followed by a linear transformation, either a linear layer or a convolution layer. Denote $\boldsymbol{x}^{(l+1)} = \mathbf{W}^{(l+1)}\boldsymbol{x}^{(l)} + \boldsymbol{b}^{(l+1)}$ be the linear transformation. Through some straightforward calculations, we have

$$\begin{aligned}\underline{\boldsymbol{x}}^{(l+1)} &= \mu^{(l+1)} - \boldsymbol{r}^{(l+1)} \\ \overline{\boldsymbol{x}}^{(l+1)} &= \mu^{(l+1)} + \boldsymbol{r}^{(l+1)}\end{aligned} \tag{19}$$

  where

$$\begin{aligned}\mu^{(l+1)} &= \frac{1}{2}\mathbf{W}^{(l+1)}(\underline{\boldsymbol{x}}^{(l)} + \overline{\boldsymbol{x}}^{(l)}) \\ \boldsymbol{r}^{(l+1)} &= \frac{1}{2}\left|\mathbf{W}^{(l+1)}\right|(\overline{\boldsymbol{x}}^{(l)} - \underline{\boldsymbol{x}}^{(l)})\end{aligned} \tag{20}$$

  Here $|\cdot|$ is the element-wise absolute value operator.

- $\boldsymbol{x}^{(l)}$ is followed by a monotonic element-wise activation function, e.g. ReLU or sigmoid. Denote $\boldsymbol{x}^{(l+1)} = \sigma(\boldsymbol{x}^{(l)})$. Then it is straightforward to see that $\underline{\boldsymbol{x}}^{(l+1)} = \sigma(\underline{\boldsymbol{x}}^{(l)})$ and $\overline{\boldsymbol{x}}^{(l+1)} = \sigma(\overline{\boldsymbol{x}}^{(l)})$.

Using the above recurrence formulas, we can calculate the interval bound of all layers $\boldsymbol{x}^{(l)}$.

Margin calculation. Finally we need to calculate the margin vector $\boldsymbol{m}$, the jth element of which is defined as the difference between neuron $x_j^{(L)}$ and $x_y^{(L)}$ where y is the target class number, i.e. $m_j = x_j^{(L)} - x_y^{(L)}$. Note that $m_y = 0$. It directly follows that if all elements of $\boldsymbol{m}$ is negative (except for $m_y = 0$) for any perturbed input, then we can guarantee robustness for data $\boldsymbol{x}$. Therefore we need to get an upper bound of $\boldsymbol{m}$, denoted as $\overline{\boldsymbol{m}}$.

One simple way to calculate $\overline{\boldsymbol{m}}$ is by using interval bound of the final layer to obtain $\overline{m}_j = \overline{x}_j^{(L)} - \underline{x}_y^{(L)}$. However, we can get a tighter bound if the final layer is a linear transformation, which is typically the case in applications. To derive a tighter bound, we first write the definition of $\boldsymbol{m}$ into a matrix form:

$$\boldsymbol{m} = \boldsymbol{x}^{(L)} - \mathbf{1}\mathbf{e}_y^T\boldsymbol{x}^{(L)} = (\mathbf{I} - \mathbf{1}\mathbf{e}_y^T)\boldsymbol{x}^{(L)}$$

where $\boldsymbol{e}_y$ is a vector with all-zero elements except for the yth element being one, $\mathbf{1}$ is the all-one vector, and $\mathbf{I}$ is the identity matrix. Note that $\boldsymbol{m}$ is a linear transformation of $\boldsymbol{x}^{(L)}$, therefore we can merge this transformation with final layer and obtain

$$\begin{aligned}\boldsymbol{m} &= (\mathbf{I} - \mathbf{1}\mathbf{e}_y^T)(\mathbf{W}^{(L)}\boldsymbol{x}^{(L-1)} + \boldsymbol{b}^{(L)}) \\ &= (\mathbf{W}^{(L)} - \mathbf{1}\mathbf{e}_y^T\mathbf{W}^{(L)})\boldsymbol{x}^{(L-1)} + (\boldsymbol{b}^{(L)} - \mathbf{1}\mathbf{e}_y^T\boldsymbol{b}^{(L)}) \\ &= \widetilde{\mathbf{W}^{(L)}}\boldsymbol{x}^{(L-1)} + \widetilde{\boldsymbol{b}^{(L)}}\end{aligned} \tag{21}$$

Using the same bounding technique in Eqn. 19,20, we can calculate $\overline{\boldsymbol{m}}$.

Loss Design. Finally, we can optimize a loss function based on $\overline{\boldsymbol{m}}$. We adopt the same loss function in Gowal et al. (2018); Zhang et al. (2020b) who use the combination of the natural loss and the worst perturbation loss. The loss function can be written as

$$l(\boldsymbol{g}, \mathcal{T}) = \frac{1}{n}\sum_{i=1}^{n} \kappa l_{\text{CE}}(\boldsymbol{g}(\boldsymbol{x}_i), y_i) + (1-\kappa) l_{\text{CE}}(\overline{\boldsymbol{m}}_i, y_i) \tag{22}$$

where l_{CE} denotes the cross-entropy loss, $\overline{\boldsymbol{m}}_i$ is defined in Eqn. 21 which is calculated using convex relaxation, and κ controls the balance between standard accuracy and robust accuracy.

As we can see, the calculation of $\overline{\boldsymbol{m}}$ is differentiable with respect to network parameters. Therefore, any gradient-based optimizer can be used to optimize these parameters. The whole process of IBP is computationally efficient: it costs roughly two times for certification compared to normal inference. However, the bound provided by IBP is looser than other more sophisticated methods based on linear relaxation.

C. Comparing ℓ_∞-dist Net with AdderNet

Recently, Chen et al. (2020) presents a novel form of networks called AdderNet, in which all convolutions are replaced with merely addition operations (calculating ℓ_1-norm). Though the two networks seem to be similar at first glance, they are different indeed.

The motivation of the two networks is different. The motivation for AdderNet is to replace multiplication with additions to reduce the computational cost by using ℓ_1-norm, while our purpose is to design robust neural networks that resist adversarial attacks by using ℓ_∞-norm.

The detailed implementations of the two networks are totally different. As shown in (Chen et al., 2020), using standard Batch Normalization is crucial to train the AdderNets successfully. However, ℓ_∞-dist Nets cannot adapt the standard batch normalization due to that it dramatically changes the Lipschitz constant of the network and hurt the robustness of the model. Furthermore, In AdderNet, the authors modified the back-propagation process and used a layer-wise adaptive learning rate to overcome optimization difficulties for training ℓ_1-norm neurons. We provide a different training strategy using mean shift normalization, smoothed approximated gradients, identity-map initialization, and ℓ_p-norm weight decay, specifically for dealing with ℓ_∞-norm.

Finally, the difference above leads to trained models with different properties. By using standard Batch Normalization, the AdderNet can be trained easily but is not robust even with respect to ℓ_1-norm perturbation. In fact, even without Batch Normalization, we still cannot provide robust guarantee for AdderNet. Remark 3.5 has shown that the Lipschitz property holds specifically for ℓ_∞-dist neurons rather than other ℓ_p-dist neurons.

D. Experimental Details

We use a single NVIDIA RTX-3090 GPU to run all these experiments. Each result of ℓ_∞-dist net in Table 1 is reported using the medien of 8 runs with the same hyper-parameters. Details of network architectures are provided in Table 5. Details of hyper-parameters are provided in Table 6. For ℓ_∞-dist Net, we use multi-class hinge loss with a threshold hyper-parameter t which depends on the robustness level ϵ. For ℓ_∞-dist Net+MLP, we use IBP loss with two hyper-parameters κ and ϵ_{train} as in Gowal et al. (2018); Zhang et al. (2020b). We use a linear warmup over ϵ_{train} in the first e_1 epoch while keeping $p = p_{\text{start}}$ fixed, increase p from p_{start} to p_{end} in the next e_2 epochs while keeping ϵ_{train} fixed, and fix $p = \infty$ in the last e_3 epochs. Different from Gowal et al. (2018), κ is kept fixed throughout training since we do not find any training instability with the fixed κ.

For other methods, the results are typically borrowed from the original paper. For IBP and CROWN-IBP results on Fashion-MNIST dataset, we use the official github repo of CROWN-IBP and perform a grid search over hyper-parameters κ and set $\epsilon_{\text{train}} = 1.1\epsilon_{\text{test}} = 0.11$. We use the largest model in their papers (denoted as DM-large) and use the learning rate and epoch schedule the same as in the MNIST dataset. We select the hyper-parameter κ with the best certified accuracy.

In Table 4, the baseline results are run using the corresponding official github repos. For example, we measure the speed for IBP and CROWN-IBP using the github repo of Zhang et al. (2020b), and measure the speed for CROWN-IBP with loss fusion using the github repo of Xu et al. (2020a).

E. Experiments for Identity-map Initialization

We conduct experiments to see the the problem of Gaussian initialization for training deep models. The results are shown in Table 7, where we train ℓ_∞-dist Nets with different number of layers on CIFAR-10 dataset, using the same hyper-parameters provided in Table 6. It is clear from the table that the *training accuracy* begins to drop when the model goes deeper. After applying identity-map initialization, the training accuracy does not drop for deep models.

F. Preliminary Results for Convolutional ℓ_∞-dist Nets

ℓ_∞-dist neuron can be easily used in convolutional neural networks, and the Lipschitz property still holds. We also conduct experiments using convolutional ℓ_∞-dist net on the CIFAR-10 dataset. We train a a eight-layer convolutional ℓ_∞-dist Net+MLP. The detailed architecture is given in Table 8. The training configurations and hyper-parameters are exactly the same as training fully-connected ℓ_∞-dist nets. Results are shown in Table 9. From Table 9, we can see that convolutional ℓ_∞-dist nets still reach good certified accuracy and outperforms all existing methods on the CIFAR-10 dataset.

Table 5. Details of network architectures. Here "Norm" denotes mean shift normalization in Section 5.1.

	ℓ_∞-dist Net (MNIST)	ℓ_∞-dist Net+MLP (MNIST)	ℓ_∞-dist Net (CIFAR-10)	ℓ_∞-dist Net+MLP (CIFAR-10)
Layer1	ℓ_∞-dist(5120)+Norm	ℓ_∞-dist(5120)+Norm	ℓ_∞-dist(5120)+Norm	ℓ_∞-dist(5120)+Norm
Layer2	ℓ_∞-dist(5120)+Norm	ℓ_∞-dist(5120)+Norm	ℓ_∞-dist(5120)+Norm	ℓ_∞-dist(5120)+Norm
Layer3	ℓ_∞-dist(5120)+Norm	ℓ_∞-dist(5120)+Norm	ℓ_∞-dist(5120)+Norm	ℓ_∞-dist(5120)+Norm
Layer4	ℓ_∞-dist(5120)+Norm	ℓ_∞-dist(5120)+Norm	ℓ_∞-dist(5120)+Norm	ℓ_∞-dist(5120)+Norm
Layer5	ℓ_∞-dist(10)	FC(512)+Tanh	ℓ_∞-dist(5120)+Norm	ℓ_∞-dist(5120)+Norm
Layer6		FC(10)	ℓ_∞-dist(10)	FC(512)+Tanh
Layer7				FC(10)

Table 6. Hyper-parameters to produce results in Table 1.

Dataset	MNIST		FashionMNIST		CIFAR-10		TinyImageNet
Architecture	ℓ_∞ Net	ℓ_∞ Net+MLP	ℓ_∞ Net	ℓ_∞ Net+MLP	ℓ_∞ Net	ℓ_∞ Net+MLP	ℓ_∞ Net+MLP
Optimizer	Adam($\beta_1 = 0.9, \beta_2 = 0.99, \epsilon = 10^{-10}$)						
Batch size	512						
Learning rate	0.02 (0.04 for TinyImageNet)						
Weight decay	0.005						
p_{start}	8						
p_{end}	1000						
ϵ_{train}	-	0.35	-	0.11	-	8.8/255	0.005
κ	-	0.5	-	0.5	-	0	0
t	0.9	-	0.45	-	80/255	-	-
e_1	50	50	50	50	100	100	100
e_2	300	300	300	300	650	650	350
e_3	50	50	50	50	50	50	50
Total epochs	400	400	400	400	800	800	500

Table 7. Performance of ℓ_∞-dist Net trained using different initialization methods on CIFAR-10 dataset.

#Layers	Gaussian Initialization		Identity Map Initialization	
	Train	Test	Train	Test
5	63.01	55.76	65.93	56.56
6	63.46	55.85	65.51	56.80
7	63.82	56.77	66.58	57.06
8	63.46	56.72	67.11	56.64
9	61.57	55.94	67.52	57.01
10	58.72	55.04	68.84	57.18

Table 8. Details of convolutional network architectures. Here "Norm" denotes mean shift normalization in Section 5.1.

	Convolutional ℓ_∞-dist Net+MLP (CIFAR-10)
Layer1	ℓ_∞-dist-conv(3, 128, kernel=3)+Norm
Layer2	ℓ_∞-dist-conv(128, 128, kernel=3)+Norm
Layer3	ℓ_∞-dist-conv(128, 256, kernel=3, stride=2)+Norm
Layer4	ℓ_∞-dist-conv(256, 256, kernel=3)+Norm
Layer5	ℓ_∞-dist-conv(256, 256, kernel=3)+Norm
Layer6	ℓ_∞-dist(512)+Norm
Layer7	FC(512)+Tanh
Layer8	FC(10)

Table 9. Performance of convolutional ℓ_∞-dist nets and existing methods.

Dataset	Method	FLOPs	Test	Robust	Certified
CIFAR-10 ($\epsilon = 8/255$)	PVT	2.4M	48.64	32.72	26.67
	DiffAI	96.3M	40.2	-	23.2
	COLT	6.9M	**51.7**	-	27.5
	IBP	151M	50.99	31.27	29.19
	CROWN-IBP	151M	45.98	34.58	33.06
	Conv ℓ_∞-dist Net+MLP	566M	49.17	**37.23**	**34.30**